2

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MIT/LCS/TR-540

REPORT ON WORKSHOP ON RESEARCH IN EXPERIMENTAL COMPUTER SCIENCE

Barbara Liskov

June 1992

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

2

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1992	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Report on Workshop on Research in Experimental Computer Science	5. FUNDING NUMBERS
6. AUTHOR(S) Liskov, B.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Massachusetts Institute of Technology Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139	8. PERFORMING ORGANIZATION REPORT NUMBER MIT/LCS/TR-540
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA 1400 Wilson Blvd. Arlington, VA 22217	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ONR N00014-91-J-4077
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE

13.
This report describes a workshop that was concerned with how to improve research in experimental computer science. The overall goal of the workshop was to identify problems and issues in experimental computer science and to propose solutions. The workshop was sponsored by the Office of Naval Research, in coordination with the NSF, DARPA, and other science agencies that participate in the Federal Coordinating Council on Science Engineering and Technology (FCCSET). It was held on October 16-18, 1991, in Palo Alto, CA.

The workshop consisted of two parts. For the first day and a quarter the entire set of attendees met as a group in an attempt to identify problems and issues that required more detailed discussion. An overview of what happened in these sessions is given in Section 1; Section 1.5 describes some conclusions based on these sessions.

The rest of the workshop was spent in small working groups that focussed on specific issues. Each of these groups was charged with coming up with a set of proposed solutioins to problems in the specific area they were addressing. Section 2 decsribes what happened in the working groups and the recommendations.

Each session (whether attended by the entire group or a subgroup) had a scribe who was responsible for taking notes during that session and providing a written summary, and a session leader who led the discussion and reported on it at the final workshop session. The summaries are included in Appendix A. Information about the program committee and attendees is contained in Appendix B.

14. SUBJECT TERMS			15. NUMBER OF PAGES 46
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE
CAMBRIDGE, MA 02139

REPORT ON WORKSHOP ON RESEARCH IN EXPERIMENTAL COMPUTER SCIENCE

October 16-18, 1991
Palo Alto, CA

Barbara Liskov

This research was supported by the Office of Naval Research under contract N00014-91-J-4077, in coordination with the NSF, DARPA, and other science agencies that participate in the Federal Coordinating Council on Science, Engineering and Technology (FCCSET).

JUNE 1992

Report on Workshop on Research in Experimental Computer Science

Barbara Liskov

This report describes a workshop that was concerned with how to improve research in experimental computer science. The overall goal of the workshop was to identify problems and issues in experimental computer science and to propose solutions. The workshop was sponsored by the Office of Naval Research, in coordination with the NSF, DARPA, and other science agencies that participate in the Federal Coordinating Council on Science, Engineering and Technology (FCCSET). It was held on October 16-18, 1991, in Palo Alto, CA.

The workshop consisted of two parts. For the first day and a quarter the entire set of attendees met as a group in an attempt to identify problems and issues that required more detailed discussion. An overview of what happened in these sessions is given in Section 1; Section 1.5 describes some conclusions based on these sessions.

The rest of the workshop was spent in small working groups that focussed on specific issues. Each of these groups was charged with coming up with a set of proposed solutions to problems in the specific area they were addressing. Section 2 describes what happened in the working groups and the recommendations.

Each session (whether attended by the entire group or a subgroup) had a scribe who was responsible for taking notes during that session and providing a written summary, and a session leader who led the discussion and reported on it at the final workshop session. The summaries are included in Appendix A. Information about the program committee and attendees is contained in Appendix B.

1 General Sessions

The purpose of the first five sessions was to identify the problems and issues that would be the focus of the breakout sessions. This section gives a brief summary of what happened in each session, and identifies certain common themes.

1.1 Opening Panel

The workshop opened with a panel featuring brief presentations by Bob Taylor, Anant Agarwal, Rick Selby, Susan Owicki, and Paul Cohen. Bob Taylor's main point was that even though the state of research in computer science as a whole has improved over the course of his career (spanning 30 years), the number of university departments capable of doing first class experimental research has increased only modestly. In addition, he stated that he believes that good experimental work requires that people build what they design and use what they build on a daily basis. Anant Agarwal's point was that we should learn to reuse one another's work and replicate one another's results. This requires credit for making one's work available to others, since the tools produced by a group may represent the group's competitive advantage, which can be reduced by early dissemination of the tools. In addition, credit for using someone else's work, support for distribution of tools, and support for the infrastructure needed to make reuse practical are also needed. Rick Selby discussed the fact that artifacts must be built and evaluated, but their construction requires a large amount of time-consuming engineering work; work can be reduced by reuse, but only well-engineered and well-supported tools are worth reusing. Susan Owicki noted that the goal of performance measurement is not just to get numbers. Instead it should identify the impact of particular approaches or techniques on performance so that we can make informed decisions on their value. Also, because technology is changing so fast, we must either get results fast, or we must abstract away from technology so that results will survive in spite of changes. Paul Cohen discussed the results of a survey of papers in AAAI-90 that indicated that AI needs more sophistication in its experimental methodology (but it was clear from the tone of the workshop discussion that this conclusion applies broadly across all fields of computer science).

1.2 Other Disciplines

In the second session we discussed experimental work outside of computer science. The first speaker was Jim Plummer; the second was Jack Owicki.

Jim Plummer works on solid state and IC applications. In Jim's field, it is very expensive to build devices, and they use extensive simulation, with very sophisticated simulation tools, prior to building. Jim believes that: all experiments measure something; all theories are wrong if you push them far enough; simulations are only as good as the models on which they are based; and adequate models for simulators always arrive too late to simulate state-of-the-art devices. Publication of research in his area is centered around small experiments that are components of large projects; a few experiments might be packaged into a thesis. The capital cost of equipment is extremely high, leading to sharing of both labs and infrastructure; even with sharing, the costs are almost more than can be borne by university faculty.

Jack Owicki is a scientist who works in a interdisciplinary area that includes chemistry, biology, and physics. He pointed out that doing experiments well involves more than just taking measurements; you must also figure out how to interpret your data. He also noted the difference between hypothesis testing and fishing expeditions (which one hopes will eventually raise hypotheses that can be tested). Model systems are useful for experimental work because they simplify reality when it gets too complicated, but they create a tension between scope and fidelity. Jack believes that the style of research differs from scientific discipline to discipline, and that no field is broadly similar to computer science although there are limited analogies in various areas.

1.3 Case Studies

The third and fourth sessions were devoted to case studies of experimental projects. The focus in the presentations was on the structure of the project and what was needed to make it work rather than on the research content of the work. In the first case studies session there were a number of short presentations; the second session featured a panel on the Spur project at UC Berkeley.

Richard Anderson described his work on implementing parallel algorithms to see how they perform in practice. He needed access to state-of-the-art multiprocessors, but he didn't use much of such a resource, so that he wanted to be able to share it with others. In addition he needed good measurement facilities, good infrastructure support (e.g., programmers and systems experts), and knowledgeable colleagues to help him understand anomalies.

Hans Berliner discussed the HiTech chess machine. This was a small project that began with a new idea of one of his students. They greatly underestimated the amount of work needed. They had support from several staff in the hardware lab (at CMU). The research was focussed on getting a particular program to work fast enough as opposed to studying specific techniques to identify their contribution to the project as a whole.

David Cheriton discussed the way that the V-system project has served as a vehicle for systems research. He believes a substantial research vehicle is needed to enable sophisticated experimental systems research. Such a vehicle brings problems to researchers' attention and allows them to tackle problems much more easily than could be done in a more conventional setting. Although many students worked on the V system simultaneously, they tended to work one-on-one with David rather than on group projects with other students. David has found that there is a tension between the work needed to maintain the environment and make progress, and the need to free students to pursue their own research objectives.

M. Satyanarayanan discussed the differences between running a project using professional staff and running one using students. (This is related to David Cheriton's last point.) Professional staff put the needs of the project ahead of individual needs to do research, and there are no problems in having them work together, and in having them reuse techniques they didn't invent instead of inventing new ones. Students must do research, have problems with joint work (in deciding what goes into whose thesis), and must have something original of their own. In addition, they are not as appreciative of simplicity as the professional staff, and their time is fractured because of the need to take classes and exams. This means their results will come along slowly so that they need to choose problems that are farther into the future.

Hank Levy discussed systems research at the University of Washington, which is unusual among academic institutions for the large amount of collaboration among faculty. Joint work is encouraged for the students; all papers have multiple authors and the content of individual theses is worked out in a dynamic and flexible fashion. New projects build upon the artifacts developed for older ones. The research methodology was: choose a problem; design a solution; choose the most efficient implementation path to demonstrate that your solution works; analyze your results and iterate until done. Furthermore, quick turnaround in being able to build a prototype is important (6 months is acceptable, 3 years isn't).

The Spur project at UC Berkeley was discussed by panelists Randy Katz and Dave Patterson (the two PIs on the project) and Susan Eggers, Jim Larus, and David Wood (project students who have gone on to faculty positions elsewhere). The project was very broad and was intended from the beginning to lead to a high-performance, working system. Key positive aspects included: the breadth led to students who were educated broadly and understood systems rather than just isolated components; going all the way to working prototypes led to more knowledge in some areas than would have been possible via functional analysis; students got broad exposure on the outside and valuable experience (the project retreats were considered particularly valuable; these were semi-annual mini-conferences in which the students made presentations as part of an external review process). However, the students felt that they spent additional time (1 to 2.5 years) because of the need to complete the overall project. (The use of professional staff was vital here, since they provide continuity, have skills students lack, free students to concentrate on research, and suffer less from the personal/group tension.) Things might have been faster if the deliverables had been ideas rather than systems, and there might have been more chance to investigate alternative designs. None of the former students is doing research on such a large scale at present, although this may come with time.

When commercial RISC machines became available (partway through the project) it might have been desirable to redirect the project away from the goal of producing a deliverable system. However, doing so would have caused problems; e.g., some students' research was contingent on continuing on the original path.

1.4 The Role of Simulation

The fifth session featured a talk by Doug Clark on simulation vs. experimentation. Doug's main point was that simulation should be considered the primary tool for evaluation of a design, and hardware should be built only as a last resort. Simulation has a number of advantages: it's cheap, it provides results quickly, you can evaluate many different samples within a benchmark suite, your results can be easily replicated, it's flexible, and you can measure any aspect of the system. But it also has disadvantages: the simulation will always be slower than the real machine, and too slow for users to experiment with it; the workloads you run may be too small (because of the speed); it may not be accurate enough; it might be wrong. Building may still be essential to overcome these problems; in addition, you must build something if you want to sell it, and students need to learn how to build boards.

Simulations will allow you to get your results sooner than really building something provided fabrication is time-consuming. If it's the design that takes most of the time, which is the case in many areas of computer science, simulation may be less helpful.

1.5 Common Themes

A number of common themes came up during the plenary sessions. First, support is needed for a number of activities that aid research but are not themselves research (at least by today's standards). We need help in building artifacts that are sufficiently robust that others can use them with confidence. We need to find a way to encourage replication of results. We need infrastructure (shared equipment, professional staff, etc.) and money to purchase tools. Support includes both funding, and ways to give certain activities (e.g., replication) academic respectability.

Second, it is clear that we have much to learn about doing performance measurements. Too often the numbers themselves are the goal, but numbers alone don't tell you enough. We need to be able to evaluate and compare techniques, and we need to do this in a way that isolates or neutralizes as many of the non-critical aspects as possible.

Third, there are two distinct styles of experimental research. Often we experiment to prove or disprove an hypothesis. Sometimes we do exploration instead. It would be a mistake to rule out exploratory research on the grounds that it isn't scientific enough. Experimental projects differ in other ways too, e.g., in scale, or in how "real" the result must be.

Fourth, the primary job of students is learning how to do research, as opposed to doing research, or building/maintaining/distributing systems. They need individual projects and unique solutions. There are many calls on their time, so progress is slower than it could be. At the least, projects of any size need professional staff, not just to provide support for infrastructure, but to help build artifacts.

Fifth, simulation is a valuable tool but it has a number of problems. Basically, a simulation runs too slowly to provide an artifact that others can use; the lack of such an artifact means that unexpected uses cannot be explored. In addition, simulations are models, which may be incorrect, or may omit details that will be important if the system is really built. Finally, for certain areas it may be as fast to build the system as to simulate it.

Finally, it is unclear how far to go when building a system. To provide a tool with good performance requires tremendous effort, more than seems justified (in many cases) to achieve research objectives. However, good performance is sometimes required to achieve other goals, e.g., adoption or technology transfer.

2 Breakout Sessions and Recommendations

This section describes the six breakout sessions. The recommendations of these sessions constitute the major output of the workshop.

2.1 Benchmarking, Measuring, and Comparing

The session on benchmarking, measuring, and comparing was concerned with techniques for evaluating the performance of systems. The group distinguished between benchmarks, which are a way of comparing the performance of systems, and workloads, which are a way of evaluating how a particular system performs.

This session came up with four recommendations:

1. There should be incentives, funding, and professional recognition for creating and disseminating benchmarks and instrumentation tools. Journals should have a section in which papers on such things as new workload-gathering tools could be published.

2. Benchmarks need to be discarded periodically and replaced with new ones so that we can avoid the problems of systems that are optimized to work well on particular benchmarks.

3. We need better methodologies for building and understanding benchmarks. In particular we need scalable benchmarks that allow extrapolation from short runs to larger systems and interpolation from long runs to intermediate points.

4. Reports on benchmarks need to be presented in enough detail to permit reproduction by those skilled in the state of the art.

2.2 Industry-University Collaboration

The session on encouraging industry-university collaboration discussed what works and what doesn't in encouraging university-industry collaboration and technology transfer, and what changes would facilitate such collaboration. It also discussed the differences in time frames for projects undertaken in industry and in academia.

The panel came up with suggestions of ways to improve collaboration:

1. The reward system of academia should reward technology transfer to industry and take it into account when making promotion decisions.

2. The reward system in industry should encourage people from industry to spend time working in academia with research groups. The duration of these visits should be at least a year or two. Many attendees at the meeting felt that US companies did not value university interaction as much as they should and that such visits were a way to improve things and to enhance technology transfer.

3. Industry is more likely to use robust technology that has been stress tested in conditions similar to that faced in industry. To encourage academics to carry through their research to this stage, funding should be provided for hiring technical staff members in universities whose job is to transform research prototypes into plausible models for industry. In addition, funding is needed to support the prototypes (or to transfer their support to an industrial organization).

The group felt that universities are the right place to do long term research, and that since such projects are risky, the right source of funds for them is government agencies. They recommended that researchers structure their work to produce interesting and demonstrable artifacts along the way to make sure that their results remain relevant in spite of technological shifts. Such artifacts should be specified as milestones in planning the research.

2.3 Experimental Methodology

The session on experimental methodology was charged with coming up with suggestions to improve the quality of experimental work. The group classified research projects based on whether the research goals were known or unknown when a project started, and whether the measurements to be taken were well understood. The categorization is useful because it affects the experimental methodology. For example, the choice of benchmark is important for projects in which both the goals and the measurements are well-understood (e.g., RISC architectures fall into this category), less important when the goals are well understood but the measurements aren't (e.g., software engineering), and largely irrelevant when neither goals nor measurements are well understood (e.g., the Arpa net in the early days).

The group came up with recommendations concerning funding, increasing the validity of experimental results, and education:

1. Funding agencies should sponsor the development of good quality workloads for different applications. Examples include workloads for integer and floating point intensive computation, database, graphics, speech, and signal processing applications. In addition to workloads, instrumentation tools should be developed, so that new workloads can easily be generated as requirements change.

2. Create sections in journals, analogous to the Correspondence section in IEEE Transactions on Computers, specifically for validating others' work. The importance of articles in this section would be less than that of regular articles.

3. Reviews for funding agencies should include a specific category for work whose primary purpose was to validate other work, so that the proposal would be reviewed in the proper light.

4. Proposals to do experimental research should have detailed sections on the methodology and the criteria that would constitute success of the experiment.

5. This latter point should apply to the reviewing process as well. Empirical papers submitted to conferences and journals should contain enough methodological details so that the work is enabling to other researchers.

6. Create a collection of great examples of experimental research in each field of computer science, in essence developing a paper role model for others' to follow. These papers should be summarized in a survey paper and also compiled into book form.

7. A curriculum should be developed for a course in experimental methods and statistics for computer science research, and offered in our departments.

2.4 Infrastructure and Funding

The session on infrastructure and funding focussed on how projects should be organized to maximize results and what form funding should take. This group discussed various forms of infrastructure including support staff, hardware, and software artifacts; such infrastructure might be shared within a single large research group, among many groups at a university, or even nationally.

This session came up with a proposal to fund a small number of broadly based research institutes at universities. This proposal was an attempt to recreate the atmosphere (and hopefully the productivity) of the big computer science labs (e.g., Project Mac). An institute would receive a large amount of money with relatively little direction on how it would be spent; most likely it would span universities and would focus on work in a particular area. Something like this has been tried in Canada. (This is the Canadian Institute for Advanced Research.) An institute would free people from having to write proposals so frequently; attendees felt that too much time was being consumed with this activity. Also, it would free them to follow new paths. However, there was disagreement about whether such an institute would be a good idea; some attendees believed that the money would be wasted and that funding specific, more directed proposals (perhaps very big ones) would be better.

A related proposal was to pair strong with weak institutions (for example, as part of an institute). This might be a way of improving the quality of weaker institutions. Some attendees felt that the CER grants had not been as effective as had been hoped (although there have been some conspicuous successes), and that the adoption plan might work better. There was broad agreement that the goal now is not to increase the number of PhDs, but rather to increase their quality.

In addition, the panel proposed a number of other action items:

1. Funding of infrastructure is a good way to leverage good systems people (who are a major scarce resource).

2. Funding of entry-level people is a good way of increasing the number of good experimentalists.

3. A variety of funding models is needed (e.g., big and small projects).

4. Competing proposals should be funded.

5. Funding agencies should be careful not to micro-manage research. A particular concern was the current emphasis in some funding agencies on dividing up a project area into subparts and assigning the subparts to individual institutions. This is not a sensible approach to doing research, and is likely to be time consuming and to not lead to a good result.

6. Care must be taken not to have standards stifle research. For example, Mach is a good platform for applications being built today, but research is needed into the platforms of tomorrow.

7. Care must be taken not to treat universities as development organizations.

2.5 Theory and Practice

The session on theory and practice discussed ways to increase the interaction between researchers in these two areas. The group agreed that such interaction was useful: theoreticians benefit by discovering interesting problems to work on, and experimentalists obtain useful "products" such as algorithms and impossibility results.

The group suggested a number of approaches for fostering interaction between theory and practice:

1. The best paper from a systems conference should be presented at a theory conference, in the hopes that it will trigger theoretical research. The dual proposal,

presenting a top theory paper at a systems conference, generated interest but was less clearly supported.

2. Ask funding agencies to support collaboration between theoreticians and practitioners. Proposals for joint work should be encouraged. Another possible organization is to have theoreticians act as consultants on system projects.

3. Encourage teaching and use of engineering analysis in computer science. Engineering analysis involves the careful use of approximations at each step. It can be contrasted to the more common pattern for computer science theory, in which an initial large approximation is made in creating the abstract model of a problem. Subsequent analysis is precise and rigorous, but the problems that are important to practitioners may be lost in the initial abstraction.

4. Identify good examples of fruitful interaction between theory and practice.

5. Encourage experimentalists to propose simple models that can be tractable for theoreticians. This can motivate theoreticians to work in an area as well as making their results more relevant to practical problems.

6. Find ways to evaluate those whose work spans more than one area. University promotion policies encourages researchers to do deep work in a single field. The former Bell Labs ranking system is an intriguing contrast, although it isn't clear how it could be applied in other settings. Until recently, all researchers in the laboratory were ranked in a single list. This was accomplished by a sort of merge sort: each line manager ranked her own people, then the lists were merged to give a ranking for the next level in the hierarchy. Since each manager could be expected to support her own people, moving ahead in the sort requires support from other managers. Such a system strongly encourages interdisciplinary work.

2.6 Large-Scale Systems and Experimentation

The session on large-scale systems and experimentation was concerned with finding ways to increase the effectiveness of research in this area. The group identified two reasons for undertaking large-scale projects: the bigness itself may be the research, or a large-scale system is needed to enable further research. They agreed that good systems people are the scarcest resource and looked for ways to increase their numbers and enhance their effectiveness.

The panel identified five important ways to attain leverage in large-scale systems research.

1. Shared Infrastructure. Sharing a common infrastructure was identified as the most important way to attain leverage; rather than building everything from the ground up, systems should build on top off a common base. To effectively share infrastructure, subareas need to clearly identify what constitutes infrastructure and what is really sharable. Mechanisms are needed to produce, distribute, and support this shared infrastructure.

2. Interfaces. Standard interfaces can be a source of tremendous leverage. Conversely, standards can also unnecessarily constrain research, especially when they are imposed by the funding agencies. Subareas need to clearly identify which interfaces make sense to standardize and which to leave unspecified.

3. To enhance sharing we need to make tools widely available. Such tools will be more useful if they are parameterizable, e.g., a parallelizing compiler that allows experimentation with optimization techniques.

4. Cooperation with Industry. Cooperating with industry, particularly in hardware prototyping projects, is almost a necessity. Most universities, and even some industrial research labs, lack the resources and engineering expertise to develop large complex systems. Identifying the limitations of university researchers and exploiting the talents of industry is key to the success of large hardware projects.

5. Leverage off of Previous Work. Large-scale projects must build upon previous work as much as possible, focusing their intellectual efforts and resources on the novel aspects of their systems. This includes building upon both previous research projects and commercial systems.

6. Raise Level of Abstraction Whenever Possible. Researchers should reduce the scale of the research project by raising the level of abstraction whenever possible. For example, the SPUR designers evaluated a large collection of instruction set extensions using instruction-level simulations. Many of the possible extensions were eliminated without considering the lower levels of abstraction (e.g., logic and circuit design).

They also identified the following problems: First, a way to evaluate people who work on large collaborative projects is needed. Second, very often the infrastructure we advocate sharing is the competitive advantage of the research group that developed it. Mechanisms are needed to encourage and reward researchers for sharing and supporting the infrastructure. Questions that need to be addressed are: How long should a group have exclusive access? How can the funding agencies encourage and reward sharing? How can universities encourage and reward sharing?

A Detailed Description of the Workshop

A.1 Opening Session

Leader: Barbara Liskov, Scribe: David Gifford

The overall goal of the workshop is to identify problems and issues in experimental computer science and to propose solutions. The outcome of the workshop will be a report of recommendations to be published and to be provided to funding agencies. Some questions we might address include:

- What are good experimental results?
- How can we do experimental research successfully?
- Should we have more standards involvement (OSI, X/OPEN, etc.)?
- How can we collaborate successfully?
- What support is needed?

The morning session began with a panel that included Bob Taylor (DEC SRC), Anant Agarwal (MIT), Rick Selby (USC), Susan Owicki (DEC SRC), and Paul Cohen (U. Mass Amherst).

A.1.1 Bob Taylor

In reading over the workshop position papers Bob Taylor felt that during our discussions we need to use with care words like science, experiment, design, and taste because they are all important to quality.

Bob has been managing science for 30 years, including 13 years with Xerox and 7 years at DEC. He has been involved in areas that include time-sharing, AI, workstations, laser printing, and personal distributed computing. All of the work he has done can be considered systems projects, but the projects have included both theoretical and experimental components. From his point of view these projects have had an impact on the shape of computing today, and although the shape of computing has changed dramatically over the past several decades, he does not believe that university research has become dramatically stronger.

Bob has one major principle for managing projects: always insist on a commitment that the people in a computer system project will build what they design, and use what they build on a daily basis. This principle is based on a belief that only through the extensive use of an artifact do you truly understand the implications of your work. In his laboratory all people use their experimental systems, from secretaries to scientists. This level of commitment and use is a necessary precondition for technology transfer.

A.1.2 Anant Agarwal

Anant Agarwal suggested that we can understand how to improve our practice of experimental computer science by examining experimental physics. He pointed out some contrasts between experimental physicists and experimental computer scientists:

- Physicists build on each others' work, starting where the last left off. Computer scientists repeat much of the foundational work, and thus have a harder time building up experimental projects. We need more discipline to reuse what others have created.
- We do not replicate each others' results. A number of reasons might underlie this. First, we may not care. Second, not enough credit is assigned to replication of work.
- We do not disseminate the tools of our work, in part because of the serious cost and overhead, and in part because the tools built by a research group may represent that group's competitive advantage – for example, a novel method for collecting multiprocessor traces is a research contribution in its own right, but is also a tool that enables further research. There is a cost to the immediate dissemination of a tool that was constructed through great effort and that represents a group s competitive advantage. Funding agencies should pay attention to this.
- Computer systems, like physics experiments, need substantial infrastructure, or the infrastructure will be the experiment itself.

He concluded that funding agencies should "encourage" dissemination, and we need better infrastructure, which can be helped with more funds and more sharing.

Discussion: David Cheriton cited the recent failures of physics as a reason not to hold them up as a model, and Steve Squires cited the state of certain labs as a reason not to hold physics up as an example. Hans Berliner said that physics would not be so regular if all physicists had the ability to invent their own electrons. A discussion of the dissemination of experimental prototypes in other sciences followed, and certain people felt that in these other sciences it was easier to take advantage of others' work. Butler Lampson did not know if it really was simpler – he wondered how easy it is to get the second cell sorter or the second scanning electron microscope.

A.1.3 Rick Selby

Rick Selby observed that researchers construct software artifacts (including tools and systems) that embody a set of software principles and key ideas. In the construction of artifacts there is an inherent tradeoff between the advancement of knowledge and time consuming engineering tasks. Artifact construction can be made easier by building on the work of others. However, often there is a reluctance of researchers to use the artifacts of others because of the amount of work required, monolithic design, poor robustness, or lack of support. This leads to the unnecessary redevelopment of components, tools and infrastructure.

The empirical evaluation of artifacts is important to advance science. Artifacts help to validate or refute techniques and theories, demonstrate quantitative advantages, and provide insight into practical impact. Some of the issues involved in such empirical studies include the high cost of setting up credible experiments, the lack of agreement on measures of central concepts (such as design flexibility or error-proneness), and the limited replication of previous studies due to cost and view that replication is not at the forefront of research.

Discussion: Satya pointed out that for effective reuse of artifacts they need to be robust, and that funding agencies may want to explicitly allow for resources to be spent on support. Others pointed out that using components in the style of DEC SRC might be difficult when everyone is not in one building. Bob Taylor said that they have a link with Paris that permits that remote lab to effectively use the components built at SRC. Rick Selby suggested that labs should "support" the software they produce, and others said that academic computer science projects have a hard time supporting exported software. Rick Rashid pointed out that international distribution of software is not that difficult. David Cheriton pointed out that export via industry when it happens is very effective, and we should not overlook this path. Butler Lampson said that using external software as a starting point for modification is a fine idea, but trying to depend on existing software components as turn-key tools is a different matter. Anant Agarwal pointed out that sometimes you get back more from your external distribution than you put in. John Hennessy said that a large infrastructure is necessary to support code, and graduate students could not be forced to do this job. Bob Taylor suggested that universities could support software if they established an appropriate organization.

A.1.4 Susan Owicki

Susan Owicki began by observing that performance measurement in experimental computer systems has the goal of identifying techniques that can be used to give good performance in subsequent systems. Performance measurement is more than a score on a benchmark because one needs to understand the factors that determine the score. The desire for understanding is motivated by the fact that the future matters more than the present.

Measurement is difficult. A phenomena may be hard to observe, measurements may perturb the system, there are too many independent users, and results from different systems are hard to compare, largely because of the large number of independent variables. However, measurement is difficult in real sciences too.

Computer science has an additional problem because technology changes so quickly that results have a limited lifetime when they are relevant. Thus we need to either get results fast or get results that last.

To get results as fast as possible we can rely on approximations, build and share tools, and take advantage of our maturing disciplines. Approximations can be built up from simulation and isolated experiments.

To get results that last we need to abstract away from technology. Thus counting operations and not microseconds is the right idea.

Discussion: John Hennessy pointed out that one can cheat with simulations, which is a problem. Jon Bentley said that one can cheat in theory also. Satya said that it would be nice if hardware provided better support for precise measurements. It was pointed out that is also important to validate simulations, and to make sure they are valid in the context in which the results are intended. Rick Rashid said that the Mach Kernel debugger provides a large amount of performance information, and similar data could be extracted from application program. Anant Agarwal pointed out that one can cheat with real implementations as well as with simulations. Ed Lazowska said that more people get misled than cheat, and it is important to have enough expertise in one group to properly interpret and present results. Butler Lampson pointed out that both simulation and implementation have their place. David Cheriton suggested that bogus papers have their place also.

A.1.5 Paul Cohen

Paul Cohen presented a study of the papers in the AAAI-90 conference. He designed the study to explore the methodology of research in AI. Of the 150 papers accepted from the 700 submitted, 89% described an example; 69% provided some analysis, proved theorems, or provided complexity results; 42% described multiple trials; 30% demonstrated performance, coverage, or compared performance; 16% probed results, unexpected results, or negative results; and 17% gave hypotheses or predictions. This suggests that AI needs more sophistication in its experimental methodology.

The goals of AI are to develop models and technology to support the design and analysis of intelligent systems. Paul studied three subjects in AAAI-90 papers: the use of models, the description of a system, and the presentation of an algorithm. 25 papers had to do with models alone; 45 with both models and algorithms; 36 with algorithms alone; 1 with both models and systems; 4 with models, algorithms, and systems; 3 with algorithms and systems; and 37 with systems alone. Model centered people look at problems such as n-queens and constraint satisfaction. Systems centered people work on problems such as path planning and inflection in speech.

Paul also discussed the recent workshop on AI Methodology. The workshop had 30 attendees, with wide-ranging backgrounds. A dozen position papers, four speakers, and five working groups were presented. The working groups included generalization and replication, comparing systems, evaluating systems that depend on knowledge, evaluating integrated systems, and theoretical AI. The workshop came up with 13 recommendations that will be published. For example, one recommendation is that AI researchers should describe their goals, and not what they are doing.

The workshop was conducted with four pleas for methodology discussions: focus on specific methodological questions, recognize that different research methods are appropriate to particular research questions, postpone discussion action items such as "infiltrating" program committees in a session dedicated to that purpose; and acknowledge the spectrum of coercion, but don't let it be paralyzing because people do want guidance.

Discussion: Paul stated that a model of research may have influenced the AAAI study in favor of experimental work to some extent, but the questions were not strongly slanted. Bill Dally said that people have focused on how to do research, but we also need to explore what to do research on. We need to judge ideas on combination of methodology and significance. It was observed that in AI it is difficult to tell if results are correct.

An unpublished paper of relevance to the AI discussion is "The Experimental Study of Machine Learning", by Pat Langley (langley@ptolemy.arc.nasa.gov) and Dennis Kibler (kibler@ics.uci.edu).

The group agreed that although Paul's presentation concerned AI, the problems existed in other areas of computer science as well.

A.2 Research in Other Disciplines

Leader: Ed Lazowska, Scribe: David Gifford

This session consisted of two talks, one by an engineer, and one by a scientist, followed by a discussion.

A.2.1 Experimental Engineering Research in Universities: Solid State and IC Applications

Jim Plummer, Integrated Circuits Laboratory, Stanford University

Jim Plummer started by describing the goals of his laboratory. University IC laboratories do not build 1.2 million transistor chips like the i486. Instead, they look farther out to technology that will be needed in the next 20 to 25 years. It is difficult to actually put a new technology into production, and thus it is done by industry.

One research area Jim's lab is exploring is reducing the line widths in IC processes below what are thought to be the existing barriers of conventional devices. For example, they are exploring GeSi transistors with very small base widths, but they are hard to build. Their experimental program requires very advanced analytical techniques because they work at the atomic level, and they also apply their skills to new areas, such as micro-machining. They also make prototype chips out of new technologies, such as A/D converters.

A second area they work in is manufacturing, and here they seek to understand how to improve traditional processes. To this end they have actually constructed a new type of process machinery that will perform many process steps on a single wafer. Many papers are published on individual aspects of such a machine, but only one paper on the entire machine has been published to date. They also explore how to achieve manufacturing technology advances in areas such as ion implantation and isolation.

Their methodology for developing new technology for devices is based upon simulations (equipment, process, and device level), followed by laboratory experiments (unit processes, split lots, and complete flows). It may take 50-100 iterations to get a new technology to work. They extensively use sophisticated simulation tools.

Jim's view on experiments vs. simulation vs. theory is as follows: (1) all experiments measure something; (2) all theories are wrong if you push them far enough; (3) simulations are only as good as the models on which they are based; and (4) adequate models for simulators always arrive too late to simulate state-of-the-art research devices.

The publication of research results usually centers around small scale experiments that are easily understood by others (e.g. boron diffusion at 1100 degrees C in oxygen). Many such results might be packaged together into a thesis (e.g. boron diffusion in silicon). Complete system experimental results are rarer and usually less useful because they give too few details to be reproduced (e.g. a complete BiCMOS process flow). Most people believe the experimental data published in the field. However, interpretation of the results is often debated.

The number of universities that can do experimental work is decreasing because of the cost of facilities. At Stanford they have a 10,000 sq feet Class 100 facility that can fabricate Si and GaAs. The original capital cost of the facility was $10-15M, with an additional $10M for equipment. The operating costs of their facility total $2.85M per year and break down as follows: supplies, $700K; permanent staff (12 technicians and 2 supervisors), $700K; electricity $800K; building systems maintenance, $400K; and equipment upgrades (they pay about 25% of the cost), $250K. Of these operating costs, $1.65M is not covered by university overhead. Perhaps 10 faculty and 100 graduate students are needed to support the facility.

One consequence of the high operating cost of their experimental facility is the sharing of equipment and technical staff. They have discovered that many faculty find projects on this scale inappropriate for university research. There is also a problem locating funding for start-up costs for new faculty. They are also working with physicists who are using semiconductor technology to build detectors for dark matter.

Time to graduation for experimental students is not necessarily longer than for theory students if the problem is partitioned properly at the outset, and if research objectives can be adjusted during the degree program.

Technology transfer is accomplished in part by having industry people on site. They consciously choose things to work on that are far enough out that proprietary issues do not hinder cooperation with industry.

A.2.2 Research in Methodologies in Scientific Disciplines

Jack Owicki, Molecular Devices

Jack Owicki works in an interdisciplinary area that includes chemistry, biology, and physics. In his experience science is grubby, and much less elegant than undergraduate science courses and the media suggest. When he is in the lab, he spends 50% of his time debugging, 40% doing experiments that fail, and 10% doing worthwhile experiments. Francis Crick stated the front lines of research are always in a fog.

In Jack's view, measurement is not the same as experimentation. Numbers without interpretation is not science. Making measurements is relatively easy. Making precise accurate measurements is harder. Knowing what you know and what you don't know is the key. The hardest thing to do is making measurements that can be interpreted unambiguously.

There are two broad kinds of experimental approaches: hypothesis testing and fishing expeditions. Fishing expeditions gather information rather than test hypotheses. They are useful because it is hard to formulate questions and hypotheses without data. At some point one must return from the fishing expedition to integrate the data with a thesis.

An experiment that seeks to test a hypothesis must be designed such that the hypothesis is falsifiable. The experiment must be able to disprove the hypothesis. A hypothesis can never be proved, just disproved. This is because there are always alternative explanations.

Model systems are useful for experimental work because they simplify reality when it is too complicated. They retain the most important features of reality (you hope), and they discard non-essential features (you hope). Model systems are at the center of the classic reductionist approach to biology. Model systems create a tension between scope and fidelity (perhaps like AI issue of coverage vs. accuracy).

Standards in scientific experimental work are of a different nature than standards in experimental computer systems. Analogues of system-performance benchmarks are not common. Science focuses on standardization to study phenomenon in standard settings, such as using standard genetically homogeneous organisms (E. coli) that are publicly available. However, standards can be constricting at times. For example, all human toxicology experiments must be done on rodents.

Reported data from experiments are widely accepted, but the interpretation that is placed on such data is subject to debate. Replication of experiments is enabled by including all relevant details in experimental publications. Replication permits others to confirm reported results and to build on a reported result. A publication should be enabling "to a practitioner skilled in the art". Replication can be inhibited if the apparatus is extremely expensive (perhaps $100M) or if a unique organism is involved. In the case of unique organisms, the principle is that they should be made available by the original experimenter to interested parties.

The style of research differs from scientific discipline to discipline. For example, physics seeks depth, uses theory extensively, and is based upon extensive collaboration because of the expensive equipment involved. Papers with 50 authors can be found, and peer review panels must allocate time on the most expensive equipment. By way of contrast, biology is not as aesthetic as physics. Biology can not always use Occam's razor because evolution did not always produce the simplest solution to biological problems. Analytical chemistry is concerned with measurements, and scientists develop measurement tools. In paleontology it is too late for direct experiments, and thus one must do imaginative analyses on the few data that are available. Small research groups result, and personnel costs dominate. In cell biology one must deal with biological complexity, and interpretations are confounded by complexity. One result is that typically many papers are required to establish an important point. Cell biology has an increasing bend toward a quantitative instrument-oriented approach. Industry-academic ties are evolving in cell biology.

In sum, no field is broadly similar to computer science, but there are limited analogies in various areas. Perhaps there are stronger analogies between experimental computer systems and older engineering disciplines. Regardless of the discipline, interpreting data is the centerpiece of the scientific enterprise.

A.2.3 Discussion:

Bob Taylor suggested that computer science might be more like mathematics, in that it is a synthetic discipline. Even though we try to understand what our artifacts are good for, it is not science. Jack Owicki pointed out that both science and computer science need to lead to new understandings and thus they are similar in important ways. Furthermore, hypothesis testing has a higher value than fishing expeditions, but at times it is hard to tell what you are doing.

In response to a question Jim Plummer said that access to equipment in his discipline is really only limited by finances because they use nonstandardized tools.

Ed Lazowska suggested that postdoctoral positions are an essential part of scientific maturation. Jack Owicki felt that the presence of postdocs in science was primarily a matter of supply and demand. Jim Plummer noted that Stanford has a new six year long terminal position (Assistant Professor of Research) that is meant to create a new kind of postgraduate career.

Bill Dally asked about how consensus is reached on the importance of topics. Jim Plummer said that there was no well accepted mechanism, but there is agreement on certain areas.

In response to a question, Jack Owicki said that the organisms studied by a biologist are self-selected. The acceptance of an organism is determined by the excitement that the papers describing it creates.

Rick Rashid asked if there was any problem getting access to tools that are present in industry but not in academia. Jim Plummer noted that students want to vary more parameters than industry, and thus their use of tools is rarely compatible with industry conventions.

A.3 Case Studies I

Leader: Marc Raibert, Scribe: Anoop Gupta

The goal of the case studies session was to get a flavor of different styles of experimental research being conducted, for example, to get a feeling for how it varies in different subareas of computer science.

A.3.1 Richard Anderson

Richard claimed to be the "token theorist" at the meeting and the focus of his talk was on *Experimental Theory*. He described his own interests as being in implementation of parallel algorithms, with the goal of narrowing the gap between theory and practice of parallel computation. His approach has been to take interesting parallel algorithms, implement them on multiprocessors (he has been concentrating on shared-memory multiprocessors), do performance measurements and analysis (understanding why theoretical predictions and observed results do not match), develop abstract models that more closely match real parallel hardware, and finally, further explore new theoretical issues raised by all of the above.

Regarding experimental style, Richard said that most projects in his group were small-scale experiments. Projects were generally quite independent and done by a single researcher. A typical project may be "Is it possible to implement this *neat* theoretical idea in practice without the overheads being too large?" or "How does the following class of algorithms perform on real machines?". Mentioning some of the issues that arise in this style of research, he said that while it was very important to have good quality input data sets for evaluating the performance of algorithms (especially for combinatorial algorithms), it was very difficult to get people excited about doing so since it is unglamorous. (I think this experience is shared by many of us in computer systems who are working on parallel processing.) He also mentioned that in this style of research it was also very important to have highly optimized sequential algorithms available, to evaluate the benefits of parallelism.

In terms of resource requirements, the key points were the following. It is very important to have access to state-of-the-art multiprocessors, since otherwise the results are not interesting. However, the hardware is needed for short periods of time to run the experiments, the implication being that the hardware can easily be shared with other research groups. Another point he stressed was the need for good measurement facilities on multiprocessors, for example, having a fine resolution timer on each of the processors. Many multiprocessors currently do not provide this. Finally, he stressed the importance and value of having knowledgeable colleagues, who understand the idiosyncrasies of the system software and hardware and can help in explaining system specific anomalous behavior.

Finally, Richard talked about the 1990-91 DIMACS implementation challenge. This was a year-long effort that culminated in a workshop from October 14-16, in which a group of 60 people participated. The goal was to implement network flow algorithms on a variety of multiprocessors. The workshop made it possible for researchers to focus on a small fixed set of problems, work with common input formats and data sets, implement lots of ideas that had been around and to compare codes, and was apparently quite successful. It will probably become an annual theory event, except that they are having difficulty getting someone to organize it for this year.

Discussion: Leo Guibas asked what was the outcome of the workshop. Richard answered that they found out what approaches were good and bad, and also what techniques worked on what classes of machines.

Jon Bentley pointed out that theoretical results may be quite different from experimental results. Richard concurred, and said that there were quite a few surprises from their implementations. He added that worst case analysis is really not very useful for evaluating the practical utility of algorithms. Richard also added that there will be a technical report published about the DIMACS workshop, that should be available soon.[1]

Barbara Liskov asked about the topics they were considering for the next workshop. Richard answered they were thinking about NP-complete optimization problems.

Dave Patterson asked what do other theorists think of experimental work and this type of workshop. Richard said it was somewhat difficult publishing such experimental work, but theoreticians were, in general, interested in it. They also look at it as a source of interesting new problems, for example, new and more realistic models of parallel machines.

A.3.2 Hans Berliner

Hans talked about the HiTech chess machine. HiTech was the highest ranked chess machine between 1985 and 1988, and the three things that were key to HiTech's success

[1]D. S. Johnson and C. C. McGeoch, editors. "DIMACS Implementation Challenge Workshop: Algorithms for Network Flows and Matching," DIMACS Technical Report 92-4, January 1992.

were: (i) its VLSI move generator, (ii) a complex evaluation function based on hardware pattern recognition, and (iii) improvements in the search technology.

Hans said that, unlike most other projects, HiTech was not a funded project at the start. It started with a new idea for a move generator from one of his theory students. At the same time Carl Ebeling, who was a hardware systems student, was looking for a chip-design project for his Area Qualifying examination. Hans put them together, and that is how the project was born. They greatly underestimated the amount of work that was needed to build the machine, but in 1985 the machine turned over. It could do search almost 500 times faster than any other program at that time, and that qualitatively changed the performance levels that could be reached. After HiTech started doing well in tournaments, it was finally put on a DARPA contract, but as research in "Massively Parallel Search" rather than on a chess machine..

As already stated, building of HiTech required a lot of work. Hans had several of his other students spend about 15% of time working on HiTech, there were several staff in the hardware lab who contributed time, and Carl put in an enormous amount of time. One of Hans' disappointments was that despite all this effort only one student got his Ph.D. out of HiTech.

Hans also described his "Ah Ha!" theory of what keeps researchers going. The theory is that you need an "Ah Ha!", or an exciting new insight, every year or two to keep you motivated and working hard. He described the theory using an anecdote about how the pattern recognizer for HiTech came into being. The realization that HiTech was losing against Master level players because of the lack of a pattern recognizer came over a weekend, by the following Tuesday Hans had worked out details of how such a recognizer could be incorporated into HiTech, and within 3 weeks Carl had built and put that hardware into the machine. The following week HiTech won most of its games against Master level players.

Discussion: There was some discussion regarding whether the HiTech work was science or engineering, but there was no clear resolution.

Pat Langley stated that the problem with chess is that there is too much focus on the end results (the game was won or lost), and that it appeared that no systematic experiments had been conducted to understand the strengths and weaknesses of HiTech. In that sense, HiTech was poor science, since what is important is that it works, but not why it works.

Berliner replied that they did do a limited set of experiments where they made a version of HiTech with limited knowledge play against the regular HiTech. He added that they would probably have done more experiments had more money been available. On the whole he was not sure if such studies would have contributed too much to knowledge.

Ed Lazowska added that an important product of the HiTech project was Carl Ebeling's thesis, and its focus was experimental VLSI design. Carl did explore various architectural tradeoffs and that was indeed a major contribution for which he got the ACM doctoral dissertation award.

Leo Guibas asked whether there were lessons from HiTech that were applicable to other areas. Hans said that HiTech showed that doing pattern matching using parallel hardware can be very effective, and this idea should probably be applicable to vision research.

A.3.3 David Cheriton

Cheriton's talk focused on the general strategy and philosophy with which he ran the V-System project. The V-System was never a specific research goal but a collection of software modules that were developed over the course of exploring a number of research issues, and exploited as a "vehicle" for on-going research. A key point was that a substantial research vehicle of this nature is required to be able to explore systems issues experimentally outside of the limitations of conventional systems. Students were generally initiated into the overall project with an initial small-scale project which focused on pushing them once through the full research cycle: build, extract results, write up, and publish/present. The objective was to establish a strong culture of results, not just hacking. The cycle generally meant that students burned in, or burned out. The research activities within the group were regarded as rather individual or student plus advisor activities, described as trying for a "stable marriage" between student and advisor, rather than a "group grope." In this vein, another part of the "culture" was to stress independent research, with acknowledgements to minor contributors, rather than group activities leading to long lists of authors on papers.

The actual definition of research activities evolved significantly over the course of the work, using what Cheriton called the "Frying Pan" approach. You attempt to build something and the computer system hits you over the head with a frying pan bringing a new problem or issue to your attention. You are then inclined to address the area of greatest pain, especially when attempting to use what you build.

Cheriton described how V had continued over a period of ten years, across 5 different research contracts, producing over 9 Ph.D. students with numerous Masters and undergraduate students participating as well. (In fact, V was derived from earlier work on Thoth and Verex, dating back further.) The key piece was the experimental software system th. brought problems to their attention and allowed them to tackle problems better than within a conventional systems framework, and faster than starting afresh after each contract.

Cheriton closed by describing his problems with providing a pioneering environment for research without making it too tough for students, and raised the question of balance between managing students and giving them free rein. The objective is to produce students who are independent researchers, but the principal investigator is a manager of a research contract and project, and needs to make progress. No easy answers were forthcoming.

A.3.4 M. Satyanarayanan

Satya talked about his experience with two distributed file system projects that he has led, the Andrew file system (AFS) and the CODA file system. The AFS was done by a team of 6 staff people (all with Ph.D.s) and CODA was done by a group of 6 graduate students. Satya decided to focus on the differences that he observed working with staff versus working with students. He made the following four observations about graduate students:

- *Staff appreciate simplicity, while students are afraid of simplicity.* Satya said that the biggest fear that a graduate student has is that somebody will say that his/her idea is simple, indirectly thinking that if it is simple, then it must not have required any deep thought or insight. He noted that staff, on the other hand, through their experience had learned the value of simplicity.

- *Search better then re-search.* Satya noted that it is very difficult to get graduate students to pursue ideas that are only subtly different from those that have existed before. A consequence can be that existing ideas may not get refined, or sometimes major weaknesses may be overlooked because nobody pursues them critically and in depth after they have been published. Satya partly attributed this problem to the fact that it is difficult to publish such work, and to get a Ph.D. for such work. Staff on the other hand, often do not have such concerns.

- *Territorialism.* Satya noted that students like to have their territory (the part of the problem that they are to solve) clearly defined and are resistant to anybody crossing into their territory and similarly do not cross into other students' territory. There are several difficulties, however. First, it is a difficult problem for the advisor, because as research progresses, areas that may have appeared quite separate earlier may not be as well separated. When this happens, which of the two students should continue on, and which one should change his/her thesis topic midstream? The students are also less willing to help out in areas that do not directly fall under their territory. Satya added that, in contrast, the loyalty of the staff is usually to the project as a whole, so such territorialism is less common.

- *Academic distraction.* He said that both faculty and students suffer from this. Basically, he believed that students and faculty were only half as productive as staff. The reason is that faculty have to teach and have numerous other responsibilities that take up their time, and students often spend considerable time taking classes and qualifying exams. A consequence is that, when working with students, one must select goals that are further into the future, which often means that the goals are less well defined.

Discussion: Brian Bershad referred to territorialism in inter-university research environments. Satya said that there the problem was even more serious. He said that the situation was not as simple as in Electrical Engineering, where things could be neatly split into searable tasks (presumably referring to Jim Plummer's talk in the morning.)

Bill Dally noted that in his experience he had found the students to be a lot more productive than the staff. The point was made that Satya was talking about a very special kind of staff that all had Ph.D.s, and that may have been the difference. Satya noted that the kind of staff he was talking about is comparable to what one would have if the research was being conducted in a good industrial research lab.

Back to the issue of territorialism, the question was raised if it was a problem if two students had common stuff in their theses. Satya said that in his experience it is not a problem as long as the overlap is small. The problem arises when down the pike one realizes that the theses will really overlap a lot. In these situations, there is no problem with staff, but it does become a problem with students.

Mendel Rosenblum wondered to what extent did the quality of the final thesis depended on a student's ability and to what extent did it depend on the portion of the work the student was (pseudo) randomly assigned at the beginning of the project. Both Dave Patterson and Randy Katz said that in their experience the student's ability was the key factor.

Rick Rashid said that he actively discouraged territorialism by actively encouraging students to look into each others code. David Cheriton disagreed saying that it was dangerous to have three students working on a topic if there was not enough stuff for three dissertations. Hans Berliner concurred with Cheriton.

John Hennessy brought up the problem of "skimming the cream". For example, one student may spend a lot of time building the base compiler (infrastructure) to try out new algorithms for compilation. Another student may totally bypass building the infrastructure, and may hand execute his novel algorithm on a few test cases and get the first publication. Worse yet, he may use the infrastructure developed by the first student to test his ideas, while the first student spends his time supporting that infrastructure. Satya added that the skimming phenomenon is a real problem in his experience too.

Bob Taylor asked if joint Ph.D. thesis were allowed for joint work. Eric Grimson said he had seen a few such Master's theses at MIT, but it was not common. He did not know of any such Ph.D. theses.

A.3.5 Hank Levy

Hank said that he would not talk about a single specific project, but rather about the general approach to experimental systems research that has been followed at the University of Washington. Referring back to some of the earlier discussions during the day, he noted that one way to *minimize* productivity is to create an environment where everyone gets credit for only and exactly what they do. He said that they have strived to avoid such an approach at Washington and that joint work, both between faculty and between students, is strongly encouraged in their environment.

Regarding the environment, he said that the key ingredients that make it special are: (i) interaction between faculty; (ii) working and collegial relationship between students and faculty; (iii) professional staff to maintain a smooth running computing environment; (iv) strong industrial ties (Hank specially cited their relationship with DEC-SRC); and finally (v) an interesting equipment base (Hank cited the Sequent, Firefly, Intel cube, and the Topaz operating system) because he said you can't do interesting research on old equipment.

Hank then went on to describe the family tree of software systems that they have developed at UW. These included Eden, Emerald, Presto, Amber, LRPC, FastThreads, Scheduler Activations, etc, which he said had had a large impact on the community. He said the key to their high productivity was that each system was based upon or extended something that they had done earlier. There was lots of sharing and building upon previous work and there were a large number of students that were working on related topics and that interacted with each other.

He next went on to talk about how the nodes in the family tree came to be. The sequence was the following. Choose a problem or an idea. Design a solution or proof of concept. Choose the *most efficient* implementation path to demonstrate that. (He noted that students most often want to take the least efficient path to demonstrate the concept.) Then measure, analyze, rethink, and repeat until satisfied with results. He said that the product is NOT the system, but rather is the research, that is, the ideas, the papers, and the students.

Discussion: Dave Patterson asked how technology transfer happened in the above model. Hank replied that it occurred mostly through papers. They had had moderate success rate in transfer of code, but they had had very high success rate in transfer of ideas.

Mark Weiser noted that Cheriton works/lives on the systems that he builds (e.g., the V system) while the Washington people don't do that, and yet both have made substantial contributions to the field. In some sense, both models appear to work.

The discussion moved back to multiple faculty and students working together, and Hank noted that he has almost no single author papers, he does not believe in them, and he hopes that he never has to write one that way.

Hank then went on to talk about the importance of having a short time to prototype. The basic idea is that if there is some idea with a moderate chance of success, then if it can be tested in a day it is highly interesting, if it is going to take 6 months it is probably still interesting, but if it is going to take 3 years it is probably no longer interesting. So the possible impact of the idea must be carefully balanced against the time it will take to prove or disprove it.

Randy Katz asked Hank if collaborative work had hampered his promotions. Hank replied that he thought that it had actually benefited him. Ed Lazowska said the key thing is to see if the person has had an impact. Rick Rashid added that coauthorship is perfectly fine and that most of his papers are that way too. Butler Lampson said that if two people can work together and produce output comparable to three, then collaboration is obviously the right thing. John Hennessy said that deans and administrative people do worry about such things so we ought to be careful about it. Many of the untenured faculty in the group listened without getting a clear picture from the senior faculty.

A.4 Case Studies II

Leader: John Hennessy, Scribe: Ed Lazowska

This session was devoted to a discussion of the Berkeley SPUR (Symbolic Processing Using RISCs) project. The participants were two of the PIs (Randy Katz and Dave Patterson), and three project graduate students who are now faculty members elsewhere (Susan Eggers, Jim Larus, and David Wood).

The Berkeley antecedents of the SPUR project were RISC-I, RISC-II, and SOAR. When the project began, RISCs had been demonstrated as chips but not as whole systems. The goals of the project were to construct a system in which C, Lisp, and multiprocessing were all first class citizens, to demonstrate that a university could build a complex hardware/software system, to produce a substantial number of well trained Ph.D.s, and to place personal multiprocessors in the hands of parallel program writers.

The project achieved all but the final goal. SPUR had 3 years of DARPA funding at roughly $1M/year beginning in 1985. (One year of effort preceded this funding.) Prototypes were booted in 1988. The system involved a shared-memory multiprocessor with 3 custom VLSI chips, the Sprite operating system, the Lisp system, 9 Ph.D. students to academia (3 of whom received NSF Presidential Young Investigator awards), and 8 Ph.D. students to industrial research. A wide variety of research was accomplished. By almost any measure, the project must be judged an outstanding success. It was not, however, a "home run": interest in Lisp waned, and RISCs and multiprocessors became commercial successes on their own merits during the course of the project.

At the workshop, the SPUR faculty and students presented a detailed and balanced view of the pros and cons of a project of this scale. Key positive aspects included:

- The integrated effort involving CAD, VLSI, architecture, operating systems, and compilers was viewed as a huge plus. Project members understand "systems" rather than isolated components.
- The project allowed convincing proof-of-concept implementations in areas such as the cache coherence protocol and in-cache address translation, and a better understanding of the overall design than would have been possible via functional analyses.
- The project contributed greatly to Berkeley's general abilities in systems areas – to a sort of "intellectual infrastructure."
- Professional benefits to the students included wide exposure, having their personal reputations enhanced by the reputation of the project, and most particularly, the project retreats. These retreats, held twice per year, had as many as 40 participants (including industrial affiliates). They provided a deadline, a source of external review, the insertion of new ideas, technology transfer, team spirit, and a forcing function for project interactions.

Suggestions for improvement were made with the acknowledgement that the project was highly successful, and that 20/20 hindsight is a huge advantage. These suggestions included:

- The students felt they spent between 1 and 2.5 years more in their Ph.D. programs than "necessary", because completion of the overall project was a goal that they embraced. The faculty were unsure of the accuracy of these estimates given that systems students at many other institutions who are not involved in such projects take as long or longer; the "time in program" of the three students was 6.5 to 7 years, which is not an inordinately large amount of time in graduate school judged by national standards. In any event, the students acknowledged that much of what they perceived of as "extra time" had a positive payoff professionally. The tension between individual and group goals was acknowledged as a difficult one to resolve.
- In this arena, the important role of professional staff was mentioned. Professional staff provide continuity, may have specific skills that student lack, free the students to concentrate on research (although it was emphasized that students must learn by doing), and suffer less from the personal/group tension.
- It was felt that the project headed towards a specific design point, without enough emphasis on the ability to investigate multiple alternative designs along the way.

- If "ideas" rather than "systems" had been the deliverables of the project, then it would have been possible to sacrifice speed for rapid development time (e.g., the Firefly) or for flexibility.

It was emphasized that the culture at Berkeley rewards collaborative research. Disincentives for this were cited by others throughout the workshop.

One key point of discussion was the choice to see the project through to completion, even though external factors made it clear that the eventual system would not be competitive. This was acknowledged by all to be a tough call. Patterson said he wished he had "the courage to change direction when commercial RISCs became available." Reasons for seeing the system through to completion included "engineering ethic," keeping promises to students, and fear of a conflict of interest since he was a consultant on SPARC, which would have been a likely candidate. And the completed system did enable a variety of good research, e.g., Wood's thesis.

A second key point of discussion was the fact that Berkeley graduates are not yet following in their advisors' footsteps in terms of the scale of experimental research that they are conducting. The SPUR graduates are clearly an "experimental" group who understand systems well. They expressed a desire to scale up their experimental efforts over the years, although all seemed leery of something on a scale as large as SPUR. It appears that the two factors currently inhibiting this scale-up are, first, relative lack of a Berkeley-style experimental culture, infrastructure, and funding at some of the departments to which they have moved - something that surely will change with time - and second, simply youth - "my ideas aren't that big yet."

A.5 Simulation vs. Experimentation

Leader: John Hennessy, Scribe: Brian Bershad

Doug Clark gave a talk about the role of simulation in computer architecture research. His main point was that simulation should be considered the primary tool with which one evaluates a design, and that the hardware should only be built as a last resort. Simulation allows the designer to find bugs in the design (design "falsification," as opposed to verification), to iteratively improve system performance, to attack a hardware problem in software while still obtaining a "prototypical" result, to filter out bad ideas before they get to silicon, and even to avoid a hardware implementation altogether. The argument here is: if all you are doing is research, then the only difference between a detailed simulation and a real system is the speed with which the two systems are able to execute code. In fact, when looking at cumulative "instruction counts," it may often be the case that one can simulate enough instructions to verify a system's design in the time that it takes a piece of hardware to return from the fab lab.

There has been a substantial increase in the number of papers over the last 15 years that have used simulation to validate design. In the 70s, most architecture papers described new "ideas" for hardware design. In the 80s, however, most of those new ideas papers also included simulations demonstrating (as best as they could) that the new ideas had some performance merit. For example, over 600 papers have appeared in ISCA from 1974 to 1991. In 1975, fewer than 1 percent used simulation. In 1990, almost 70 percent used simulation. In contrast, older papers had more real hardware.

Simulations have some advantages over real pieces of hardware which make them more attractive. First, they're cheap. An off-the-shelf processor can be used to simulate any arbitrary piece of custom silicon. Second, (and in a related vein), simulations run on standard hardware. Moreover, hardware speed is increasing so simulations can grow in complexity. Third, unlike hardware, it is easy to make a copy of a simulation. This allows you to run many instances at once, increasing the sample size of the benchmark suite being examined. Also, you can send copies to your friends, allowing them to replicate and verify your results. Fourth, it is relatively easy to fix design bugs in a simulation. Lastly, in a simulation, it is possible to measure ANY aspect of the system being simulated (more accurately, you can measure any part of the simulation; you can not generally measure anything in the system that is beneath the lowest level of the simulation.)

With simulation, you have the results soon, but not faster. This echoes a comment made earlier by Wood from Wisconsin that it is often better, in a research project, to sacrifice speed for flexibility and turnaround time.

Once hardware is designed, it is possible to begin simulating the system. It may be possible to get in enough cycles with the simulation long before you get the hardware built. The assumption here is that fabrication takes a lot of time. If only instruction count matters, this is true. Otherwise, most of the time goes into design of the low level simulator to be sure that things work. The fab time is low.

However, simulations have shortcomings. They just aren't fast enough. They don't convince the end users that your architecture is usable and useful and reasonable. Verifying the potential design requires building. We are always on the edge of what we can reasonably simulate because whatever we just built is what we are using to simulate what we want to build next, which is always more complicated than what we are running on now. A second aspect to the speed problem is that it may cause designers to run workloads which are not large enough (or long-running enough) to adequately evaluate the system. For instruction level simulators, the speed argument is becoming less valid. Machine emulation can be done with a slowdown of 2x. The compiler and operating system design for the mips, for example, was all done on the vax.

A simulation may not be accurate enough. Aspects of the design may be left out out, for example, performance aspects that might be important but which, from the standpoint of the model, do not appear to be important. Your simulator could just be broken. Simulator verification is a hard problem. It all falls down to having faith in models and implementation. Validation is based on "lots of sweat."

Simulation forces you to be overly conservative. Simulation alone forces you to avoid the leading edge because you only stay with well understood underlying technology.

Simulation alone allows you to push off, and ignore forever, nasty physical problems. In the academic world, this may result in overaggressive designs, which end up assuming that since industry can do something, then an academic project can do it. This is dangerous because industry may have to also do things that slow down performance to increase yield or reliability, for example. To assume all of the good things and not the bad ones is unfair and misleading.

The only really moral approach is to pretend that you are going to build the whole thing.

The question of when to build and when to simulate is the WRONG question because we must always simulate. A better question is when must we build?

For non-research goals, building is essential. These goals include: making money from a new hardware widget (you can only earn from what you sell, and few people want to run their programs on simulators); technology transfer (industry is not yet "savvy" enough to buy a design which has only been verified by a simulator); and grad student training (someone has to learn how to build boards).

For research projects, the process of fabrication does several things:

- it validates your design process. Did your simulator "work?"
- you build yourself an extremely fast simulator. After you've built it, it's not a final machine. It's actually a simulator in its own right IF you've built it correctly because you are able to tweak parameters and simulate in real time. This is a critical reason to BUILD!!!! Fast simulation of new ideas.
- With hardware, you have a real stimulus to get people to write code for your machine. Again, the issue is performance. You are never going to have a simulator for a machine that runs faster than the machine which you are running.

A.6 Breakout Session on Benchmarking, Measuring and Comparing

Leader: M. Satyanarayanan, Scribe: Brian Bershad

This session was structured as follows: we passed the token around the room and each person made a brief statement about what they felt was an appropriate (or inappropriate) aspect of benchmarking. From these statements (which sometimes amounted to more than one per person), a set of 4 key recommendations and three "warnings" were developed.

A.6.1 Key Recommendations

1. Rewards. There should be incentives and professional recognition for creating and disseminating benchmarks. This involves: 1) recognition that generating good benchmarks is a hard problem; 2) funding for the creation and distribution of benchmarks; and 3) academic recognition in the form of publication avenues for benchmark material.
2. Old benchmarks should fade away. Benchmarks, like forms of government, should be periodically thrown away and replaced with new ones. This ensures that "benchmark resistant" strains of systems cannot be built, or if they are built, will only remain superior for a short period of time.
3. We need better (any) methodologies for building and understanding benchmarks. The benchmark runners should be more familiar with statistics and sample interpretation. The benchmark builders must become more familiar with designing scalable benchmarks; that is, benchmarks for which short runs allow extrapolation to larger systems, and long runs allow interpolation at intermediate points. If benchmarks are to be used to evaluate hypothetical systems, then it must be possible to use

existing systems (with their limited computing power) to evaluate the performance of systems which have not yet been built. Scalable benchmarks are the only way to do this.

4. Remember the scientific method. Reports on benchmark runs should be presented in adequate detail for reproduction by "those skilled in the state of the art." This means that we need to be more accurate about describing our experimental environment.

A.6.2 Warnings

In addition to these major suggestions, three "warnings" about interpreting benchmarks arose:

- Don't cheat on benchmarks and be aware of those who do.
- Make sure that the benchmark environment is reproducible.
- Insignificant deltas in benchmark results SHOULD BE IGNORED. People need to understand that there is a "meaninglessness threshold" beneath which nothing matters.

A.6.3 Discussion:

What follows are the detailed points that led to these recommendations and warnings.

1. There are inadequate file system benchmarks.

The lack of good (any) file system benchmarks necessitates that the file system researcher creates their own. This is both a pain and likely only to represent the biases and expectations of the individual who created them.

Unlike many systems, perhaps benchmarks should only be created by committee, to ensure that several different perspectives are represented.

2. It is difficult to determine what a benchmark really captures.

Often, the creator of the benchmark may not be aware that the benchmark is actually measuring some obscure system detail (example: file system lookup performance which can be influenced by the size of the password file).

3. Systems are sometimes rigged to do well on a specific benchmark. (This is related to point 2).

Rashid relayed the story of a comparison between Ultrix and Mach in which one particular Ultrix benchmark performed hundreds of times better than the same benchmark under Mach. On looking at the benchmark, Rashid discovered that it was creating and deleting a zero length file. Under Mach, this would require several disk accesses. Under Ultrix, however, the test required no accesses because Ultrix had been "optimized" to recognize the case and do nothing.

The benchmark was SUPPOSED to measure some aspect of file system performance (exactly what aspect is not entirely clear because people rarely create and then delete zero length files), but in fact only ended up measuring a certain short path through the kernel. Ultrix was cheating.

4. Benchmarks should be thrown away periodically. Old benchmarks should be replaced with new benchmarks on a periodic basis to ensure that systems don't become "benchmark resistant." This way peculiarities and anomalies in benchmarks don't end up influencing system design for too long.

Doug Clark drew an analogy with forms of government: no matter what it is, it should be replaced every 99 years to rid the system of those who have figured out how to corrupt it.

5. There needs to be a set of mechanisms in place for publicizing and disseminating information about, and the actual, benchmarks.

There is no doubt that there are a large number of file system stress applications, parallel programs, etc, out there in the community. There is unfortunately no adequate way of distributing these benchmarks. Calls for programs often appear on the comp.* newsgroups, but these are haphazard.

As there are today with the RFPs, there should be a central (or set of central) FTP repositories with interesting programs for people to use.

6. There needs to be a hierarchy of benchmarks.

7. Problems need to be ranked.

8. Someone needs to provide a comprehensive suite of benchmarks for comparison.

A system which does "well" on a hard benchmark ought to be expected to do even better on an "easier" one. This was some disagreement what exactly this hierarchy entailed, or even if it could exist.

9. Someone has to deal with the preservation and maintenance of benchmarks.

Marketing forces imply that this "somebody" has to be from the academic side of things; more specifically, they can not be from industry because of the conflict of interests that this would create.

Presently, there is the PERFECT club from Illinois, but the people there have discovered that this is far too expensive and time consuming to maintain.

SPECmarks, which is not an industrial effort, currently requires a small fee from commercial vendors to be included, plus the devotion of one full time engineer.

To get it right, benchmark technology should be a specifically FUNDED effort, much as some of the technology transfer work (distributing software developed in a university) is being funded.

10. We need to make a distinction between BENCHMARKS and WORKLOADS. A benchmark is used to compare different systems according to some scalar number (e.g., this system is BETTER than that system because it has a higher SPECmark).

WORKLOADS are used to evaluate a given system in an attempt to tune the system.

11. As much as possible, there should be a standardized trace format. All of the comments from this discussion should therefore apply to traces.

We should be willing to consider TRACES as a flavor of a benchmark.

There has been some standardization of traces in the architecture community; there's just so many ways that you can represent loads and stores. Moreover, writing a filter for someone else's trace format into your own is not generally hard.

Things become trickier when talking about file system traces, for which the kinds of operations are much higher level than for architecture (open file, seek pointer, etc.). The existence of standard operating system interfaces should imply that we have standard trace information.

The situation gets harder when dealing with systems in which "standard paths" through the system being traced can be circumvented. For example: virtual memory traces when all you have is physical memory. Or, file system traces when all you have is mapped files and fault patterns. Auxiliary information in the trace file will be necessary.

People sometimes hoard traces. This is a bad thing. Somebody said that they thought that there was once a guy who hoarded file system and cache traces, but they couldn't remember his name.

12. Old traces may become obsolete as the interface changes. This is just a problem.

13. Proprietary benchmarks may sometimes be unavailable. This is only a problem if you are trying to evaluate new tests against proprietary tests, or if you are trying to "explain away" the results of a proprietary benchmark.

14. People need to be educated as to what benchmarks actually mean.

Benchmarks can be misleading, especially for those who are easily misled. A simple scalar which reflects integer performance may sway someone who wants to run floating point intensive applications. This person will be disappointed.

There's a bit of "lies, damn lies, and statistics" in here as well. Funny uses of means, variances, geometric means, etc., can often conceal the truth, or be used to reveal a truth that actually isn't.

15. The precise conditions under which benchmarks are run should be described when the results of the benchmarks are presented.

Example: the performance of many file system benchmarks can be influenced by the length of the password file (because of having to do name lookups on the file owners). Unless this is known to the users of the benchmark and presented to the interpreters of the benchmark the results from a file system test can be meaningless. Or, at the very least, irreproducible.

The hard part here is in understanding what factors influence the performance of a benchmark. Often times, the interactions between all of the different pieces of the system may not be known because of the "Black Box" syndrome.

16. People need to receive academic credit for creating and distributing benchmarks.

If benchmarks are hard to come by, and good benchmarks are hard to build, then it should be the case that good papers on good benchmarks should be publishable in forums like ASPLOS, ISCA and Sigmetrics. The consensus was that Sigmetrics was a particularly good place to publish.

It SHOULD NOT be the case that benchmark papers only appear in non-refereed publications like SIGOPS. This gives authors an incentive to write good papers about good suites.

17. Benchmarks have certain characteristics that lead to their wide spread use. These characteristics are:

- Portability. The benchmark can be run on many different systems without too much effort. Good examples: The Andrew file system benchmark; UNIX based SPECmarks; C based benchmarks.

- People can read the benchmarks and without a whole lot of effort determine if they are reasonable or bogus. Also, they can determine how the results they get correlate to the system which is being measured. Complex benchmarks make this very difficult.

- Realistic, that is, predictive of system performance in general. This is hard to do, and is getting much harder because machines no longer behave enough like one another to ensure that there is always going to be a positive correlation on benchmarks run across different systems. System variations now occur in: Memory (cache sizes), I/O, Cache, CPU.

18. Looking for a single number to characterize performance is BOGUS.

Clark pointed out that there was an implicit assumption that benchmarks which result in a scalar number are somehow a true predictor of machine performance, but because systems architectures are become so diverse, this is becoming less true. Only by looking at the results of a suite of programs, and by being able to identify where one's anticipated applications fit into that suite, is it possible to get any value out of benchmarks. Otherwise, benchmark suites are only able to predict the behavior of programs in the suite.

19. We need to have a scaling methodology.

It should be possible to run a smaller/larger version of a benchmark on a system and somehow correlate that to a smaller/larger system. This will allow us to run a long program in a short period of time, and to understand how changes in the SIZE of a system influence performance.

A.7 Breakout Session on Encouraging Industry-University Collaboration

Leader: John Hennessy, Scribe: Keshav Pingali

The group discussed three issues:

- what works and what does not in encouraging university-industry collaboration and technology transfer.
- changes that would facilitate such collaboration.
- differences in time frames between industry and academia.

A.7.1 What works

1. Technology transfer occurs when faculty members not only publish research papers but are committed to transferring the fruits of their research to industry.

2. It is important to have active involvement by industry people in research projects while the research is being conducted. Down-loading a project to industry after the work is complete, without having done some priming, does not work too well.

3. Having the ear of an influential person in the company is important; otherwise, the technology may not get the further investment required.

4. Although papers by themselves do not usually lead to technology transfer, pape to industry-oriented conferences are useful in facilitating it.

5. Faculty members who take sabbaticals in industry return with an appreciation of the real problems faced by industry.

6. Cost sharing by DARPA and other funding agencies is helpful in propagating ideas to industry. If the Government backs something, industry usually follows. The government can also support the explicit transfer process by providing staff to improve the quality of technology which is being transferred.

A.7.2 What does not work

1. Papers to academic conferences usually have no impact on industry, unless followed through with concrete interaction with industry.

2. Past experience demonstrates that visitors from US companies do not cause technology transfer to occur. The panel felt it was because US companies do not value university interaction as much as they ought to and do not send visitors with the ability to assist in technology transfer on a consistent basis.

A.7.3 Suggested changes

To improve industry-university collaboration, the panel made the following suggestions:

1. The reward system of academia should reward technology transfer to industry and take it into account when making promotion decisions.

2. The reward system in industry should encourage people from industry to spend time working in academia with research groups. The duration of these visits should be at least a year or two.

3. Industry is more likely to use robust technology that has been stress tested in conditions similar to that faced in industry. To encourage academics to carry through their research to this stage, funding should be provided for hiring technical staff members in universities whose job will be to transform research prototypes into plausible models for industry.

A.7.4 Time horizons

One overriding concern expressed by the panel was the 'impedance mismatch' between the time frames of academia and industry. Traditionally, industry views academic research as being too long-term, speculative and risky, while academia views industry's focus as being too myopic. To achieve common ground, the panel made the following recommendations.

1. Universities are the right place to do long term research. However, academics should view technology transfer as a long pipeline in which the products of completed projects are delivered to industry even as work gets underway on long-term projects that will not reach the industry for many years.

2. Since it is difficult to predict technology trends many years into the future, long term projects are risky and unlikely to be funded by industry. Therefore, the right source of funds for long-term work are government agencies. To make sure that this kind of work stays relevant in spite of technological shifts, researchers should structure their work to produce interesting and demonstratable artifacts along the way. Such artifacts should be specified as milestones in the process of proposing and planning the research.

A.8 Breakout Session on Experimental Methodology

Leader: Susan Eggers, Scribes: Susan Eggers, Eric Grimson

The methodology breakout session focussed on two distinct issues: (1) classifying experimental research in such a way that each category would represent a different set of problems to doing good research, and therefore a different set of solutions and (2) devising a list of concrete solutions to improve the quality of experimental research.

The categorization was necessary because the problems faced by researchers in experimental work differ across subdisciplines within computer science. It hinges on whether the research goals are initially known or unknown and whether the measurements are well defined or vague. In the first category both goals and measurements are well understood when the research is being done. This criteria applies to research that develops new solutions to known problems, whether the solutions are radical or incremental. In this case the goals are often concrete and the measurements are those that had been applied in the past. The example given was RISC architectures. In the second category the goals are known, but the measurements are vague either because they are poorly understood or require experiments that are too large. Software engineering fits both categories, because it is difficult to measure questions such as whether a system is easy to use or maintain, and the test of the system is often its use by hundreds of users. The Arpanet is an example of a research project in which both the goals and measurements were unknown when the project was envisioned. Many experiments, such as those involving parallel architecture , fall into more than one category. On the one hand, its speedup experiments are well defined; on the other, the novelty of the architecture may lend itself to being used in ways unenvisioned when it was designed. An example of using the categorization is that the choice of benchmark is important for the first category (known, well-defined), less important for the second (known, vague) and irrelevant for the third (both unknown).

We also discussed such issues as:

- how to design realistic test cases;
- how to avoid measuring only what you know how to measure;
- how to avoid workload-driven research;
- the role of repositories of workloads;
- whether we need more data on the state of experimental research, using the Cohen and Clark studies as models;
- the differences between exploratory and experimental research.

The latter half of the breakout focussed on devising a list of concrete suggestions to improve several methodological aspects of doing experimental research. The first was a request to funding agencies to sponsor the development of good quality workloads for different applications. Examples include workloads for integer and floating point intensive computation, database, graphics, speech and signal processing applications. In addition to workloads, instrumentation tools should be developed, so that new workloads can easily be generated as requirements change. Both suggestions would facilitate empirical research, since the overhead of creating a valid and representative workload would be removed.

The second group of suggestions addressed the lack of validation of empirical work. An often heard comment from the workshop at large was that, unlike our colleagues in the physical sciences, researchers in computer science made little or no attempt to validate the empirical research of others. The sentiment was that this was a failing in our methodology and should be addressed. The goal of all suggestions was to create a forum for doing research validation and a climate in which replicating others' work was valued. The goal of replicating should be either to confirm the previous work, dispute it, or test it under a different set of criteria. The suggestions were:

- Create sections in journals, analogous to the Correspondence section in IEEE Transactions on Computers, specifically for validating others' work. The importance of articles in this section would be less than that of regular articles.
- Reviews for funding agencies should include a specific category for work whose primary purpose was to validate other work, so that the proposal would be reviewed in the proper light.
- Proposals to do experimental research should have detailed sections on the methodology and the criteria that would constitute success of the experiment.
- This latter point should apply to the reviewing process as well. Empirical papers submitted to conferences and journals should contain enough methodological details so that the work is enabling to other researchers.

Discussion in the session at large objected to getting publications credit for simply replicating research, without any additional results, such as disputing or building on the previous work.

The last group of suggestions addressed the more educational aspects of doing good empirical research. The first was to create a collection of great examples of experimental research in each field of computer science, in essence developing a paper role model for others' to follow. These papers should be summarized in a survey paper and also compiled into book form. Secondly, a curriculum should be developed for a course in experimental methods and statistics for computer science research, and offered in our departments.

A.9 Breakout Session on Infrastructure and Funding

Leader: Jim Larus, Scribe: Eric Grimson

We began with a general discussion that included the following points:

1. What is right size of funding in experimental work? Does it vary from one area to another?
2. Where on spectrum should one partition funds? e.g. McArthur approach versus old boy networks, NSF versus DARPA.
3. What is the right way to maximize research? e.g. argument that DARPA wants to define tasks and interfaces and distribute them among sites, against the idea of support for multiple approaches to same problem.
4. General discussion about too much micro-management by places like DARPA.
5. Be careful not to confuse standards with infrastructure, i.e., have to be able to explore standards for the future.
6. Does one even want to allow for such bureaucracy in projects (argument that such models have frequently had problems in the past).
7. Strong argument made that want to have competing projects.
8. Need to separate out use of a project from role as next generation of work in an area, e.g., MACH is a good platform but should not be allowed to inhibit development of competing models.
9. Need to discuss issues of separating mandating of platform from enabling of platform.

Next, there was a discussion of forms of infrastructure:

1. Producing artifacts for community, e.g., experimental vehicle/base, toolkit versus platform base.
2. Staff and culture (experience and continuity).
3. Hardware/software/networks (utilities for community).
4. Support for acquisition of artifacts.

5. Service (MOSIS) – national infrastructure.
6. Shared data/benchmarks.

There was some discussion on need to set up Institutes for systems research (roughly on the $15 million level). These would be mandated to be high level with no micromanagement. There was some discussion of how to set them up, e.g. individual universities versus distributed set of sites. (Something like this has been done in Canada and it worked well.) One advantage is that it would free people from needing to write proposals so often.

There was discussion about the issue of industry/university collaboration. It hasn't worked well in the past; the question is why?

There was also discussion of problems with peer review process.

Finally, the discussion turned to Actions Items:

1. Institutes.
2. Funding of infrastructure as way of leveraging good people (who are major scarce resource).
3. Entry level funding.
4. Need variety of funding models.
5. No micro-management.
6. Don't treat universities as development houses.

A.10 Breakout Session on Theory and Practice

Leader: Jon Bentley, Scribe: Susan Owicki

Participants in this session agreed from the start that interaction between theory and practice is valuable to both. For theory, the benefit is in new problems and models from which to conduct theoretical analysis. For practice, the benefit comes both from the products of theory – algorithms, impossibility results, and the like – and from the simplicity, elegance, and better communication that a clear understanding promotes.

However, in spite of these benefits there is far too little connection between the two. We discussed a number of obstacles that limit interaction, identified some instances of fruitful interrelation, and proposed approaches to fostering cross-fertilization of ideas. These are discussed below.

Our definition of theory was quite broad. We included the traditional theory community (the folks that go to FOCS and STOC) as well as formal work in such fields as queueing networks, AI, and security. In addition, we included less formal heuristic approaches for understanding and reasoning about aspects of computer systems. Even when the theory is informal, it is important to be explicit about assumptions, chains of reasoning, expected results, and how the assumptions can be validated.

We didn't feel the need to define practice; the implicit assumption seemed to be that "practice" meant building hardware or software.

There are obstacles to theory/practice interactions in each of the disciplines. Within practice, the major difficulty is that some practitioners see little value in theory. It was suggested that some major conferences, particularly SOSP and SIGARCH, frequently reject papers that base their results on analytic models. This may reflect a lack of familiarity with the analytic tools that makes it hard for practitioners to evaluate the results.

Within theory, the major difficulty is that theoreticians have little motivation to make their work applicable to practice. This operates first at the level of problem selection: theory work is valued more for elegance, mathematical sophistication, and contribution to the development of theory than for its relevance to practical problems. And once the work is done, theoreticians often do not take the effort to explain the results in ways that are accessible to practitioners.

In addition to these obstacles, there are problems inherent in interdisciplinary work. Often it requires people with expertise in both the application and in theory; such people are hard to come by. This is especially true because career advancement depends most on the opinion of people in one's own specialty. At tenure time, at least one set of references must feel that the candidate's work has made important contributions to their field. Finally, attempts at joint conferences tend to be perceived as theory conferences, and before long they are. Instances of this phenomenon were mentioned in computational geometry and parallel algorithms/architectures. It was also observed that researchers may not send their best work to joint conferences.

In spite of these obstacles, there have been many instances of profitable collaboration between theory and practice. The group identified several examples, including language theory and compiler front ends, work by Bentley and others on implementing bin packing algorithms, and Leiserson's circuit retiming techniques.

Finally, the group considered what a number of approaches for fostering interaction. Suggestions included:

1. The best paper from a systems conference should be presented at a theory conference, in the hopes that it will trigger theoretical research. The dual proposal, presenting a top theory paper at a systems conference, generated interest but was less clearly supported.

2. Ask funding agencies to encourage joint work and theoretical consultants on systems projects.

3. Encourage teaching and use of engineering analysis in computer science. Engineering analysis involves the careful use of approximations at each step. It can be contrasted to the more common pattern for computer science theory, in which an initial large approximation is made in creating the abstract model of a problem. Subsequent analysis is precise and rigorous, but the problems that are important to practitioners may be lost in the initial abstraction.

4. Identify good examples of fruitful interaction between theory and practice.

5. Encourage experimentalists to propose simple models that can be tractable for theoreticians. This can motivate theoreticians to work in an area as well as making their results more relevant to practical problems.

6. Find ways to evaluate those whose work spans more than one area. University promotion policies encourages researchers to do deep work in a single field. The former Bell Labs ranking system is an intriguing contrast, although it isn't clear how it could be

applied in other settings. Until recently, all researchers in the laboratory were ranked in a single list. This was accomplished by a sort of merge sort: each line manager ranked her own people, then the lists were merged to give a ranking for the next level in the hierarchy. Since each manager could be expected to support her own people, moving ahead in the sort requires support from other managers. Such a system strongly encourages interdisciplinary work.

A.11 Breakout Session on Large-Scale Systems and Experimentation

Leader: Anoop Gupta, Scribe: David Wood

In this session we discussed issues in large-scale systems research, the respective roles of universities and industry, and tools and techniques that can improve the success of these projects. Funding and infrastructure were explicitly excluded from consideration.

A.11.1 Reasons for Large-Scale Systems Research: When and Why?

We first asked the question "when and why should large-scale systems (LSS) research be undertaken?" Large-scale projects are sometimes begun simply because they are large, even though a smaller-scale project would suffice. Sometimes this occurs because large infrastructures can take on a life of their own, causing researchers to select new large-scale projects simply to maintain the infrastructure. Alternatively, researchers sometimes undertake large-scale projects that replicate significant portions of previous work either because they are unable or unwilling to build upon it. As a group, we agreed that research projects should generally be no larger than necessary to achieve the research goals.

We identified three major styles of research in large-scale systems. The first two require large-scale research projects to build large-scale systems, while the third style calls for smaller-scale projects that use or modify existing large-scale systems.

1. *The Large-Scale System is Research Itself*
In this style of research, the system does not exist and significant research is required to build it. The goals of the system are often broad and ill-defined; the research is usually more exploratory than quantitative. Examples of this type of research are: the Connection Machine, Smalltalk, and the J-Machine.

2. *Large-Scale System is an Enabling Technology*
In this style of research the large-scale system does not exist and must be built to *enable* further research. The technology required to build the large-scale system itself is fairly well understood, and the bulk of the research comes from using the large-scale system, not building it. Examples are the DEC SRC Firefly, CommonLisp, and Berkeley UNIX.

3. *Refinement of Large-Scale Systems*
In this style of research, a large-scale system already exists somewhere and the researcher(s) would like to build upon it or modify it. The key issue here is access to the existing large-scale system; without access the researcher may be forced to unnecessarily replicate previous work. Examples are compilers and database

systems. Studying a new compiler optimization technique should only require modifying a high-quality optimizing compiler, rather than developing a new compiler from scratch.

An important point is that the research style in a particular subarea generally evolves over time. Initially the system is research itself, then becomes an enabling technology, and finally, when the subarea becomes more mature, allows quantitative and iterative evaluation of the system.

A.11.2 How to Improve Large-Scale Systems Research

We then addressed the question of how to improve large-scale systems research. The group agreed that large-scale projects are often difficult to complete, and tried to identify key ways of improving the effectiveness of the research. The general consensus was that *good large-scale systems people* are the most scarce resource. The key goals should be to

1. Maximize the effectiveness and impact of good people
2. To increase their numbers over time

The key to achieving the first objective is *leverage.*

We identified 5 important ways to attain leverage in large-scale systems research, and selected one good example for each.

1. *Shared Infrastructure*
   Sharing a common infrastructure was identified as the *most* important way to attain leverage; rather than building everything from the ground up, systems should build on top off a common base. An obvious example is parallel compiler research: each research group should not have to develop a system from scratch. Instead, a production-quality parallelizing compiler should be made available to all members of the research community. Other examples are simulators, address traces, and parallel applications.

   To effectively share infrastructure, subareas need to *clearly* identify what constitutes infrastructure and what is really sharable. Mechanisms are needed to produce, distribute, and support this shared infrastructure.

2. *Interfaces*
   Standard interfaces *can* be a source of tremendous leverage. Continuing with the compiler example, a carefully specified compiler intermediate form can greatly increase the ability of researchers to leverage off of one another's work. Conversely, standards can also unnecessarily constrain research, especially when they are imposed by the funding agencies. Subareas need to clearly identify which interfaces make sense to standardize and which to leave unspecified.

3. *Cooperation with Industry*
   Cooperating with industry, particularly in hardware prototyping projects, is almost a necessity. Most universities, and even some industrial research labs, lack the resources and engineering expertise to develop large complex systems. Identifying

the limitations of university researchers and exploiting the talents of industry is key to the success of large hardware projects. A good example is the MIT/LCS Logic/Sun collaboration to develop the SPARCLE processor chip. Industry provided the engineering and manufacturing expertise while MIT provided the innovative design.

4. *Leverage off of Previous Work*
   Large-scale projects must build upon previous work as much as possible, focusing their intellectual efforts and resources on the novel aspects of their systems. This includes building upon both previous research projects and commercial systems. For example, the Stanford DASH project is built by adding boards to Silicon Graphics multiprocessor workstations, leveraging off of their bus-based cache-coherence mechanism.

5. *Raise Level of Abstraction Whenever Possible*
   Finally, researchers should reduce the scale of the research project by raising the level of abstraction whenever possible. For example, the SPUR designers evaluated a large collection of instruction set extensions using instruction-level simulations. Many of the possible extensions were eliminated without considering the lower levels of abstraction (e.g., logic and circuit design).

A.11.3 Open Questions

Very often the infrastructure we advocate sharing is the *competitive advantage* of the research group that developed it. Mechanisms are needed to encourage and reward researchers for sharing and supporting the infrastructure. Questions that we identified but did *not* address are:

1. How long should a group have exclusive access?
2. How can the funding agencies encourage and reward sharing?
3. How can universities encourage and reward sharing?

B Attendees at the Workshop

Program Committee

Barbara Liskov, Chairperson
Jaime Carbonell
John Hennessy
Edward Lazowska
Marc Raibert

Attendees

Anant Agarwal
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139
(617) 253-1448
agarwal@mit.edu

Richard Anderson
Dept. of Computer Science and Engr.
University of Washington
Seattle, WA 98195
(206) 543-4305
anderson@cs.washington.edu

Jon Bentley
AT&T Bell Laboratories
Room 2C-317
Murray Hill, NJ 07974
(908) 582-2315
jlb@research.att.com

Hans Berliner
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-2577
berliner@k.cs.cmu.edu

Brian Bershad
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-3041
Brian.Bershad@cs.cmu.edu

Guy Blelloch
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-6245
guy.blelloch@cs.cmu.edu

David Cheriton
Bldg. 460, Rm. 424
Stanford University
Stanford, CA 94305
(415) 723-1131
cheriton@pescadero.stanford.edu

Douglas Clark
Digital Equipment Corp.
85 Swanson Rd. (BXB1-2/H11)
Boxboro, MA 01719
(508) 264-5555
clark@crl.dec.com

Paul R. Cohen
Dept. of Computer Science, A309
University of Massachusetts
Amherst, MA 01003
(413) 545-3638
cohen@cs.umass.edu

William J. Dally
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139
(617) 253-6043
billd@ai.mit.edu

Richard A. DeMillo
Purdue University
Dept. of Computer Science
W. Lafayette, IN 47907-1398
(317) 494-7823
rad@cs.purdue.edu

Susan Eggers
Dept. of Computer Science and Engr.
University of Washington
Seattle, WA 98195
(206) 543-2118
eggers@cs.washington.edu

John Gannon
Dept. of Computer Science
University of Maryland
College Park, MD 20742
(301) 405-2671
gannon@cs.umd.edu

David Gifford
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139
(617) 253-6039
gifford@mit.edu

James R. Goodman
University of Wisconsin-Madison
Computer Sciences Dept.
Madison, WI 53706
(608) 262-0765
goodman@cs.wisc.edu

Eric Grimson
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge MA 02139
(617) 253-5346
welg@ai.mit.edu

Thomas Gross
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213
(412) 268-7661
thomas.gross@cs.cmu.edu

Leonidas J. Guibas
Computer Science Dept.
Stanford University
Stanford, CA. 94305
(415) 723-0304
guibas@cs.stanford.edu

Anoop Gupta
CIS-212
Stanford University
Stanford CA 94305
(415) 725-3716
ag@pepper.stanford.edu

Harry Hedges
National Science Foundation
1800 G St. NW
Washington D.C. 20550
hhedges@nsf.gov

John Hennessy
CIS 208
Stanford University
Stanford, CA 94305
(415) 725-3712
jlh@vsop.stanford.edu

Randy H. Katz
University of California
Computer Sciences Division - EECS
Berkeley, CA 94720
(510) 642-8778
randy@cs.berkeley.edu

Gary M. Koob
Office of Naval Research
Computer Science Division (Code 1133)
800 N. Quincy St.
Arlington VA 22217-5000
(703) 696-0872
koob@itd.nrl.navy.mil

Butler Lampson
DEC Cambridge Research Laboratory
1 Kendall Square
Cambridge, MA 02139
(617) 621-6619
lampson@crl.dec.com

Pat Langley
AI Research Branch - Mail Stop 244-17
NASA Ames Research Center
Moffett Field, CA 94035
(415) 604-4878
langley@ptolemy.arc.nasa.gov

James Larus
Computer Sciences Dept.
University of Wisconsin-Madison
Madison, WI 53711
(608) 262-9519
larus@cs.wisc.edu

Edward D. Lazowska
Dept. of Computer Science and Engr.
University of Washington
Seattle, WA 98195
(206) 543-4755
lazowska@cs.washington.edu

Hank Levy
Dept. of Computer Science and Engr.
University of Washington
Seattle, WA 98195
(206) 543-9204
levy@cs.washington.edu

Barbara Liskov
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139
(617) 253-5886
liskov@lcs.mit.edu

Matt Mason
Carnegie Mellon University
School of Computer Science
Pittsburgh, PA 15213
(412) 268-8804
mason@cs.cmu.edu

Jack Owicki
University of California
105A Donner Laboratory
Berkeley, CA 94720
(510) 642-8412
owicki@garnet.berkeley.edu

Susan Owicki
DEC Systems Research Center
130 Lytton Ave.
Palo Alto CA 94301
(415) 853-2270
owicki@src.dec.com

David A. Patterson
Computer Science Division - EECS
University of California
Berkeley, CA 94720
(510) 642-0930
pattrsn@cs.berkeley.edu

Keshav Pingali
Dept. of Computer Science
Cornell University
Ithaca, NY 14853
(607) 255-7203
pingali@cs.cornell.edu

Jim Plummer
CIS-114
Stanford University
Stanford, CA 94305
(415) 725-3606
plummer@sierra.stanford.edu

Bruce Porter
Dept. of Computer Sciences
University of Texas
Austin, TX 78712
(512) 471-9565
porter@cs.utexas.edu

Marc Raibert
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139
(617) 253-2478
mxr@ai.mit.edu

Richard F. Rashid
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052
(206)936-3432
rashid@microsoft.com

Mendel Rosenblum
Computer Sciences Division - EECS
University of California
Berkeley, CA 94720
(510) 642-9669
mendel@cs.berkeley.edu

Gary Sabot
Thinking Machines Corp.
245 First St.
Cambridge MA 02142
(617) 234-2836
gary@think.com

M. Satyanarayanan
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-3743
satya@cs.cmu.edu

Richard W. Selby
Dept. of Information & Computer Sci.
University of California
Irvine, CA 92717
(714) 856-6326
selby@ics.uci.edu

Robert F. Sproull
Sun Microsystems Laboratories
2 Federal St.
Billerica, MA 01821
(508) 671-0353
rsproull@east.sun.com

Stephen L. Squires
DARPA/SISTO
3701 N. Fairfax Drive
Arlington, VA 22203-1714
(703) 696-2226
squires@darpa.mil

Eric Sumner
AT&T Bell Labs
Naperville IL 60566
(708) 713 7660
ees@research.att.com

Robert Taylor
DEC Systems Research Center
130 Lytton Ave.
Palo Alto, CA 94301
(415) 853-2100
taylor@src.dec.com

Alex Waibel
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-7676
waibel@cs.cmu.edu

Mark Weiser
Computer Science Laboratory
Xerox PARC
3333 Coyote Hill Rd.
Palo Alto, CA 94304
(415) 494-4406
weiser@xerox.com

Robert Wilensky
University of California
Computer Sciences Division - EECS
Berkeley, CA 94720
(510) 642-7034
wilensky@larch.berkeley.edu

David A. Wood
University of Wisconsin
Computer Sciences Dept.
Madison, WI 53706
(608) 263 7463
david@cs.wisc.edu

DARPA OFFICIAL DISTRIBUTION LIST

DIRECTOR Information Processing Techniques Office Defense Advanced Research Projects Agency (DARPA) 1400 Wilson Boulevard Arlington, VA 22209	2 copies
OFFICE OF NAVAL RESEARCH 800 North Quincy Street Arlington, VA 22217 Attn: Dr. Gary Koop, Code 433	2 copies
DIRECTOR, CODE 2627 Naval Research Laboratory Washington, DC 20375	6 copies
DEFENSE TECHNICAL INFORMATION CENTER Cameron Station Alexandria, VA 22314	2 copies
NATIONAL SCIENCE FOUNDATION Office of Computing Activities 1800 G. Street, N.W. Washington, DC 20550 Attn: Program Director	2 copies
HEAD, CODE 38 Research Department Naval Weapons Center China Lake, CA 93555	1 copy